# Introduction to SQL

Class 3

# Course Overview

- Introduction to SQL
    - Databases, Tables
    - Classification of SQL – DDL, DML, DCL, TCL
        - DDL – CREATE, ALTER, DROP
        - DML – SELECT, INSERT, UPDATE, DELETE
        - DCL – GRANT, REVOKE
        - TCL – COMMIT, ROLLBACK, SAVEPOINT
    - Data types, Operators
    - Keys – Primary, Foreign, Composite, Unique, Alternate
    - Integrity Constraints – Domain Integrity Constrains, Entity Integrity Constraints, Referential Integrity Constraints
    - Joins – Outer Joins, Left Outer Joins, Right Outer Joins, Inner Joins.
    - Queries, Subqueries, Functions, Flow Control (IF, CASE, WHILE, FOR, LOOP), Stored routines
    - Views
    - Concurrency and locking (Implicit locks, explicit locks, row level locks, table level locks, database level locks)
    - Indexes, Cursors, Triggers, Events
    - Tuning SQL queries and optimizing performance
    - SQL Databases vs NoSQL Databases
    - ACID, CAP
    - How SQL databases internally works

## NULL values

- Used to represent missing or unknown data
- It is different from an empty string or a zero value because it indicates that the value is unknown or is not applicable rather than explicitly set to a particular value
- We use IS NULL or IS NOT NULL to check if a column has null values

```
SELECT *
FROM employees
WHERE phone_number IS NULL;
```

- While comparing a column or an expression where the LHS is a column
  - NULL!=NULL false
  - NULL=NULL false
  - NULL IS NULL true
  - NULL IS NOT NULL false
- It is not included in COUNT(column_name), because it does not mean anything

# SQL Operators (Can be used in MySQL as well)

- Reserved words and characters that can be used with WHERE clause in SQL Query
- There are two types of operators
  - Unary Operator
    - Syntax
      - Operator SQL_Operand
  - Binary Operator
    - Syntax
      - Operand1 SQL_Operator Operand2

# SQL operators are categorized in the following categories:

1. SQL Arithmetic Operators
2. SQL Comparison Operators
3. SQL Logical Operators
4. SQL Set Operators
5. SQL Bit-wise Operators
6. SQL Unary Operators

## SQL Arithmetic Operators

1. SQL Addition Operator (+)
2. SQL Subtraction Operator (-)
3. SQL Multiplication Operator (*)
4. SQL Division Operator (/)
5. SQL Modulus Operator (%)

## SQL Comparison Operators

1. SQL Equal Operator (=)
2. SQL Not Equal Operator (!=)
3. SQL Greater Than Operator (>)
4. SQL Greater Than Equals to Operator (>=)
5. SQL Less Than Operator (<)
6. SQL Less Than Equals to Operator (<=)

## SQL Logical Operators

1. SQL ALL operator
2. SQL AND operator
3. SQL OR operator
4. SQL BETWEEN operator
5. SQL IN operator
6. SQL NOT operator
7. SQL ANY operator
8. SQL LIKE operator

# SQL ALL operator

- The ALL operator in SQL compares the specified value to all the values of a column from the sub-query in the SQL database.
- It can be used with the following statements
  - SELECT
  - HAVING
  - WHERE
- Syntax
  - SELECT column_Name1, ...., column_NameN FROM table_Name WHERE column Co mparison_operator ALL (SELECT column FROM tablename2)
  - SELECT Emp_Id, Emp_Name FROM Employee_details WHERE Emp_Salary > ALL ( SEL ECT Emp_Salary FROM  Employee_details WHERE Emp_City = Jaipur)

## SQL Between Operator

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1 AND value2;
```

It is similar to using >= AND <= operators combined.

## SQL IN Operator

- The **IN operator** in SQL allows database users to specify two or more values in a WHERE clause.
- This logical operator minimizes the requirement of multiple OR conditions.
- Syntax
  - SELECT column_Name1, column_Name2 ...., column_NameN FROM table_Name WHERE column_name IN (list_of_values);
  - SELECT * FROM Employee_details WHERE Emp_Id IN (202, 204, 205);
  - SELECT * FROM Employee_details WHERE Emp_Id NOT IN (202,205);

## SQL NOT Operator

- The **NOT operator** in SQL shows the record from the table if the condition evaluates to false.
- It is always used with the WHERE clause.
- Syntax
  - SELECT column1, column2 ...., columnN FROM table_Name WHERE NOT condition;
  - SELECT * FROM Employee_details WHERE NOT Emp_City = 'Delhi' ;
  - SELECT * FROM Employee_details WHERE NOT Emp_City = 'Delhi' AND NOT Emp_City = 'Chandigarh';

# SQL ANY Operator

- The **ANY operator** in SQL shows the records when any of the values returned by the sub-query meet the condition.
- The ANY logical operator must match at least one record in the inner query and must be preceded by any SQL comparison operator.
- **Syntax**
  - SELECT column1, column2 ...., columnN FROM table_Name WHERE column_name comparison_operator ANY ( SELECT column_name FROM table_name WHERE condition(s)) ;

# SQL LIKE Operator

- The **LIKE operator** in SQL shows those records from the table which match with the given pattern specified in the sub-query.
- The percentage (%) sign is a wildcard which is used in conjunction with this logical operator.
- This operator is used in the WHERE clause with the following three statements:
  1. SELECT statement
  2. UPDATE statement
  3. DELETE statement
- Syntax
  - SELECT column_Name1, column_Name2 ...., column_NameN FROM table_Name WHERE column_name LIKE pattern;
  - SELECT * FROM Employee_details WHERE Emp_Name LIKE 's%' ;
  - SELECT * FROM Employee_details WHERE Emp_Name LIKE '%y' ;
  - SELECT * FROM Employee_details WHERE Emp_Name LIKE 'S%y' ;

## SQL Set Operators

1. SQL Union Operator
2. SQL Union ALL Operator
3. SQL Intersect Operator
4. SQL Minus Operator (Not present in MySQL)

## SQL Union Operator

- Combines the result of two or more SELECT statements and provides the single output
- The data type and the number of columns must be the same for each SELECT statement used with the UNION operator
- Syntax:

```
SELECT column1, column2 ...., columnN FROM table_Name1 [WHERE conditions]
UNION
SELECT column1, column2 ...., columnN FROM table_Name2 [WHERE conditions];
```

- Duplicates are removed here

## What if there are two rows which have one column same and one column different

- The UNION operator will treat those two rows as separate, unique rows and include both of them in the result set.

# SQL Union ALL Operator

- The SQL Union Operator is the same as the UNION operator, but the only difference is that it also shows the same record.
- Syntax

```
SELECT column1, column2 ...., columnN FROM table_Name1 [WHERE conditions]
UNION ALL
SELECT column1, column2 ...., columnN FROM table_Name2 [WHERE conditions];
```

- Duplicates are not removed

## SQL Intersect Operator

- Shows the common record from two or more SELECT statements.
- The data type and the number of columns must be the same for each SELECT statement used with the INTERSECT operator
- Syntax

```
SELECT column1, column2 ...., columnN FROM table_Name1 [WHERE conditions]
INTERSECT
SELECT column1, column2 ...., columnN FROM table_Name2 [WHERE conditions];
```

## SQL Minus Operator

- Used to retrieve all rows from the first SELECT statement that are not present in the second SELECT statement.
- In other words, it returns the difference between the results of two SELECT statements
- The number of columns and the data types in each column must be the same for both SELECT statements.
- Syntax

```
SELECT column1, column2, ... FROM table1
MINUS
SELECT column1, column2, ... FROM table2;
```

- MINUS is not supported by Mysql. We achieve this kind of behaviour by using NOT EXISTS

## SQL Unary Operators

1. SQL Unary Positive Operator
2. SQL Unary Negative Operator
3. SQL Unary Bitwise NOT Operator

## SQL Unary Positive Operator

- Makes the numeric value of the SQL table positive
- Syntax
  - SELECT +(column1), +(column2) ...., +(columnN) FROM table_Name [WHERE conditions] ;

## SQL Unary Negative Operator

- Makes the numeric value of the SQL table negative.
- Syntax
  - SELECT -(column_Name1), -(column_Name2) ...., -(column_NameN) FROM table_Name [WHERE conditions] ;

# SQL Unary Bitwise NOT Operator

- Provides the one's complement of the single numeric operand.
- If the bit of any numerical value is 001100, then this operator turns these bits into 110011.
- Syntax
  - SELECT ~(column1), ~(column2) ...., ~(columnN) FROM table_Name [WHERE conditions] ;

## SQL Bitwise Operators

The Bitwise Operators in SQL perform the bit operations on the Integer values

1. Bitwise AND (&)
2. Bitwise OR(|)

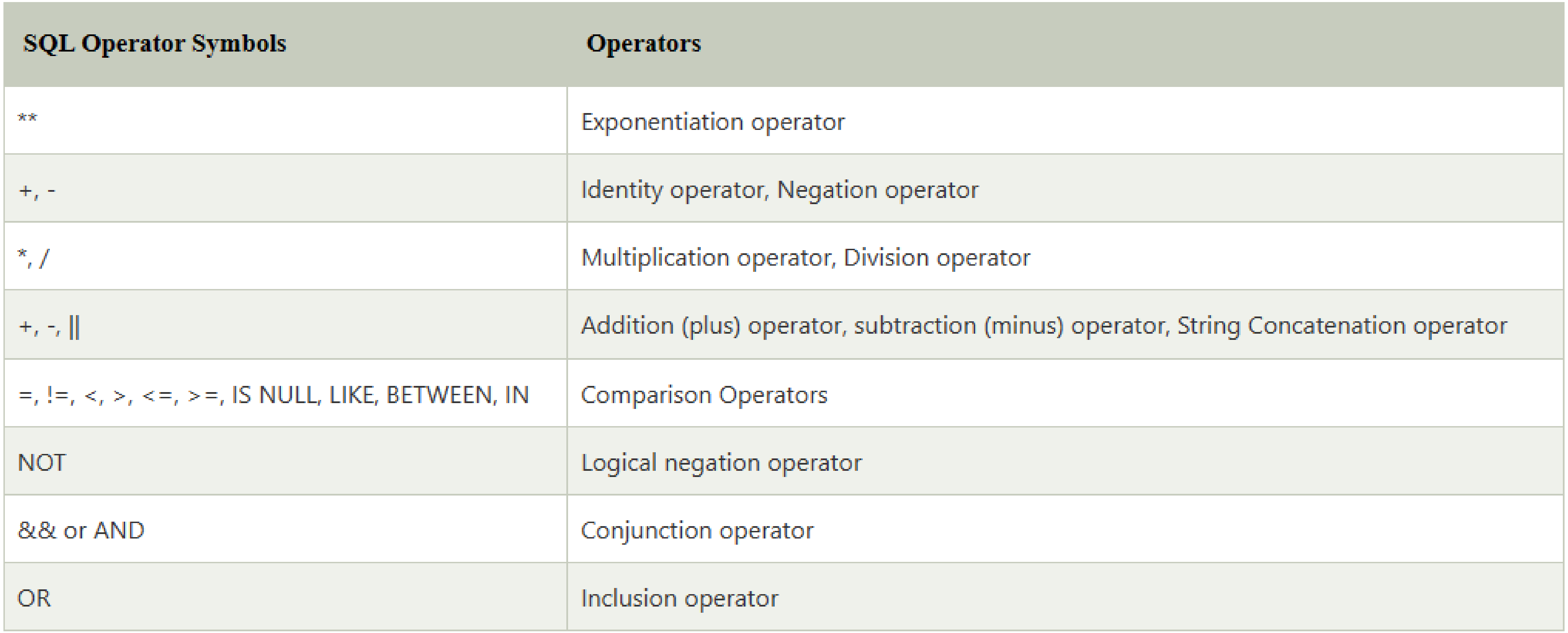
## Bitwise AND (&)

- Performs the logical AND operation on the given Integer values.
- This operator checks each bit of a value with the corresponding bit of another value.
- Syntax
  - SELECT column1 & column2 & .... & columnN FROM table_Name [WHERE con ditions] ;
- When we use the Bitwise AND operator in SQL, then SQL converts the values of both columns in binary format, and the AND operation is performed on the converted bits.

# Bitwise OR (|)

- Performs the logical OR operation on the given Integer values
- Syntax
  - SELECT column1 | column2 | .... | columnN FROM table_Name [WHERE conditions] ;
- When we used the Bitwise OR operator in SQL, then SQL converts the values of both columns in binary format, and the OR operation is performed on the binary bits. After that, SQL converts the resultant binary bits into user understandable format, i.e., decimal format.

## Operator precedence



| SQL Operator Symbols | Operators |
|---|---|
| ** | Exponentiation operator |
| +, - | Identity operator, Negation operator |
| *, / | Multiplication operator, Division operator |
| +, -, \|\| | Addition (plus) operator, subtraction (minus) operator, String Concatenation operator |
| =, !=, <, >, <=, >=, IS NULL, LIKE, BETWEEN, IN | Comparison Operators |
| NOT | Logical negation operator |
| && or AND | Conjunction operator |
| OR | Inclusion operator |

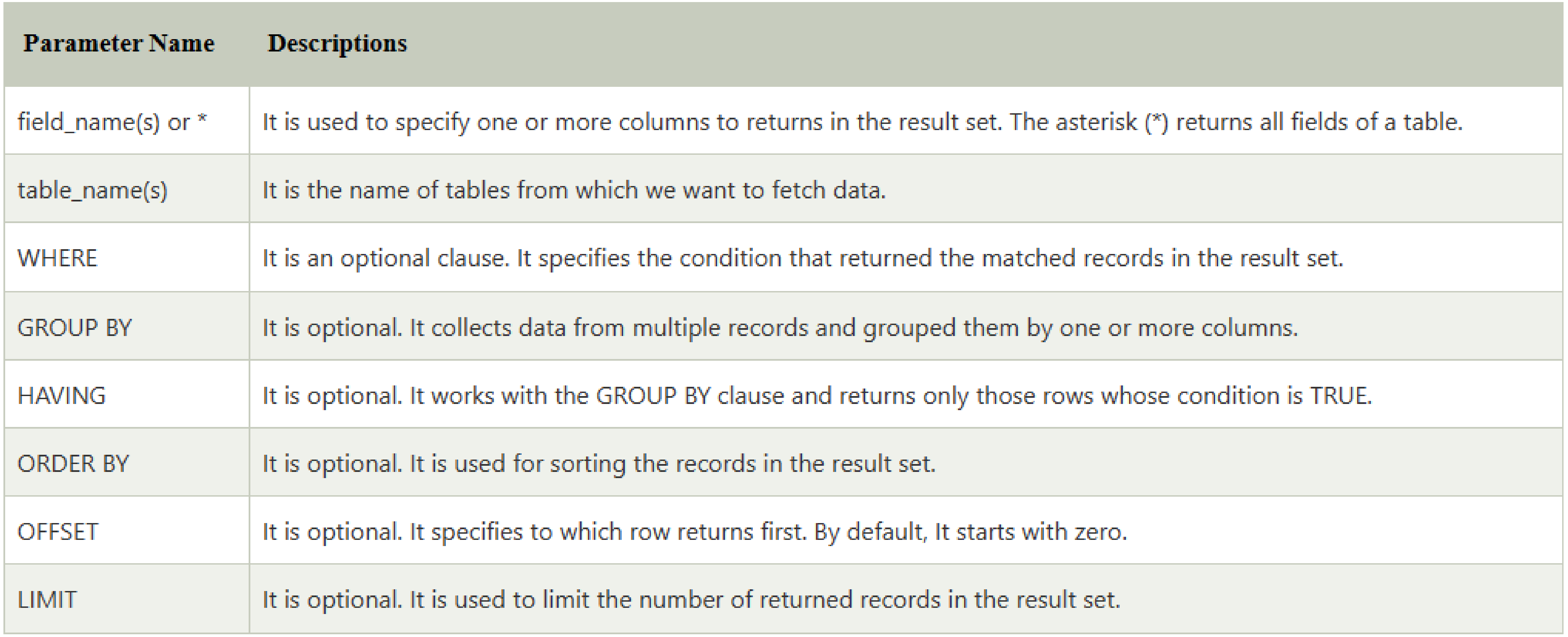
## LIMIT and OFFSET (Pagination)

- Limit is how many number of rows you want from the output
- SELECT * FROM table_name LIMIT [Number to Limit By];

- Offset is from where you want to start returning the data
- Or you can also say how many rows you want to skip before starting to return the results
- SELECT * FROM artists LIMIT [number of rows you want to show] OFFSET [Number of rows to skip];

## Final Select Query

```
SELECT field_name1, field_name 2,... field_nameN
FROM table_name1, table_name2...
[WHERE condition]
[GROUP BY field_name(s)]
[HAVING condition]
[ORDER BY field_name(s)]
[OFFSET M ][LIMIT N];
```



| Parameter Name | Descriptions |
|---|---|
| field_name(s) or * | It is used to specify one or more columns to returns in the result set. The asterisk (*) returns all fields of a table. |
| table_name(s) | It is the name of tables from which we want to fetch data. |
| WHERE | It is an optional clause. It specifies the condition that returned the matched records in the result set. |
| GROUP BY | It is optional. It collects data from multiple records and grouped them by one or more columns. |
| HAVING | It is optional. It works with the GROUP BY clause and returns only those rows whose condition is TRUE. |
| ORDER BY | It is optional. It is used for sorting the records in the result set. |
| OFFSET | It is optional. It specifies to which row returns first. By default, It starts with zero. |
| LIMIT | It is optional. It is used to limit the number of returned records in the result set. |

Connect with me on LinkedIn

Please subscribe to our YouTube channel

Check out my GitHub profile

Follow me on Twitter(X)

## Thank you